```python
import time
import threading
import numpy as np
from music21 import converter, stream, instrument, note, chord
from transformers import Text2Speech # □□□□□□□□□□□

# ========== □□□□□□□□□ ==========
class RobotActuator:
def __init__(self, instrument_type):
self.instrument_type = instrument_type

def play_note(self, midi_num, velocity, duration):
"""□□□□□/□□□□□□"""
print(f"[{self.instrument_type}] Playing MIDI {midi_num} with velocity
{velocity} for {duration}s")
# □□□□□□□□□□ ROS □□□□□□
# self.ros_publish(midi_num, duration)

# ========== □□□□□□□□ ==========
class SingerRobot:
def __init__(self):
self.tts = Text2Speech(model="vocaloid-style") # □□□□□□□□□

def sing(self, lyrics, melody):
"""□□□□□□□"""
audio = self.tts.generate(lyrics, pitch_curve=melody)
# audio.play() # □□□□□□□
print(f"Singing: {lyrics}")

# ========== AI □□□□ ==========
class ConductorAI:
def __init__(self, band, singer):
self.band = band
self.singer = singer
self.tempo = 120 # BPM

def load_score(self, musicxml_path):
"""□□□□□□□MusicXML □□□"""
self.score = converter.parse(musicxml_path)

def start_performance(self):
"""□□□□□□□"""
threads = []
for part in self.score.parts:
if part.partName == 'Vocals':
self._start_vocal_thread(part)
```

```
else:
threads.append(self._start_instrument_thread(part))

for t in threads:
t.join()

def _start_instrument_thread(self, part):
"""□□□□□□"""
def play_part():
robot = next(r for r in self.band if r.instrument_type == part.partName)
for event in part.flat.notesAndRests:
if isinstance(event, note.Note):
robot.play_note(event.pitch.midi, 64,
event.duration.quarterLength*(60/self.tempo))
elif isinstance(event, chord.Chord):
# □□□□□□□□□□□□□□□□□□
for p in event.pitches:
robot.play_note(p.midi, 64, event.duration.quarterLength*(60/self.tempo))
return threading.Thread(target=play_part).start()

def _start_vocal_thread(self, part):
"""□□□□"""
lyrics = ["Jingle bells, jingle bells...", "Should auld acquaintance be forgot..."] # □
□□□
melody = [60, 62, 64, ...] # □□□□
self.singer.sing(lyrics, melody)

# ========== □□□□□ ==========
if __name__ == "__main__":
# □□□□□□
band = [
RobotActuator("Piano"),
RobotActuator("Violin"),
RobotActuator("Cello"),
RobotActuator("Saxophone"),
RobotActuator("Trumpet")
]
singer = SingerRobot()

# AI □□□□□□
conductor = ConductorAI(band, singer)
conductor.load_score("scores/jingle_bells.xml") # □□□□□ MusicXML □□
conductor.tempo = 150 # □□□□□□

# □□□□
conductor.start_performance()
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

● [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

4. Multi-robot collaborative performance
In the band scene, multiple robots need to play together. This requires designing a master control system, which is responsible for coordinating the rhythm and melody of each robot to ensure the harmony of the whole performance.

Example of implementation steps

1. Music score parsing: parsing music score files (such as MIDI files) into data structures containing note information.

2. Parameter conversion: the note information is converted into robot control parameters, such as key position, duration and strength.

3. Control instruction generation: According to the control parameters, generate control instructions that the robot can execute, such as pulse signals or motor driving instructions.

4. Signal receiving and processing: write a signal receiving module to process the performance signal and control the robot to enter the corresponding state.

5. Collaborative performance control: In the multi-robot scene, design a master control system to coordinate the performance actions of each robot.

Through the above steps, you can write corresponding playing instructions for different types of robots and realize their playing of various musical instruments.
● It is extremely complicated to realize the whole program code of such an artificial intelligence big model multi-model multi-modal general agent high intelligent robot holding a concert, which involves the technical integration in many fields. The following is just a very simple example of conceptual Python code framework (the reality is far from enough, and it needs to be combined with a lot of hardware and more complicated algorithms, etc., here is just a schematic), mainly involving music playback simulation and other parts:

```
python

Import playsound # is used to play audio, and this library needs to be installed first: pip install playsound.

# Define a function to play music
def play_music(music_path):
try:
playsound.playsound(music_path)
except:
Print(f "unable to play music: {music_path}")

# Assume the audio file paths of different instruments and singing.
```

```
Bell_path = "Jingle bells.mp3" # needs to be replaced with the actual file path.
Auld Lang Syne _path = "Auld Lang Syne.mp3" # Need to be replaced with the
actual file path.
Piano_path = "Piano playing fragment.mp3" # needs to be replaced with the
actual file path.
Violin_path = "violin playing fragment.mp3" # needs to be replaced with the
actual file path.
Cello_path = "cello performance fragment.mp3" # needs to be replaced with the
actual file path.
Saxophone_path = "saxophone playing fragment.mp3" # needs to be replaced
with the actual file path.
French_horn_path = "French trumpet playing fragment.mp3" # needs to be
replaced with the actual file path.
Male_singer_path = "Male singer's singing clip.mp3" # needs to be replaced with
the actual file path.
Female_singer_path = "Female singer singing clip.mp3" # needs to be replaced
with the actual file path.

# Simulate the conductor's beat (simple print tips here)
def conduct():
Print ("The conductor starts to beat, and the concert begins!" )

# Simulate the performance and singing process
def perform():
conduct()
Print ("Start playing Jingle Bells")
play_music(bell_path)
Print ("piano playing part")
play_music(piano_path)
Print ("violin playing part")
play_music(violin_path)
Print ("cello playing part")
play_music(cello_path)
Print ("saxophone playing part")
play_music(saxophone_path)
Print ("French trumpet playing part")
play_music(french_horn_path)
Print ("male singer singing part")
play_music(male_singer_path)
Print ("female singer singing part")
play_music(female_singer_path)
Print ("Start playing Auld Lang Syne")
Play_music (auld lang syne _path)
Print ("At the end of the concert, the audience applauded warmly!" )

if __name__ == "__main__":
perform()
```

The above code is only a very basic simulation. The actual intelligent robot concert needs to involve many complex technologies and system integration, such as robot control (including mechanical arm control of musical instrument operation, etc.), multi-modal interaction (such as visual recognition of audience response, etc.), audio synthesis and processing (synchronization of different musical instruments and singing, etc.), and intelligent decision-making (such as adjusting performance according to field conditions, etc.).

● To realize the concert completed by multi-modal general agent and robot band, it is necessary to integrate core technologies such as artificial intelligence large model, multi-modal interaction, real-time audio and video technology and robot motion control. The following is a summary of the design and technical realization path of the core module (the complete code needs to be combined with the specific hardware interface and development framework, and the technical framework and key code logic are provided here):

First, the system architecture design

```
# Overall Architecture Example
class ConcertSystem:
def __init__(self):
# Core module initialization
Self.music _ model = multimodal music model () # multimodal music generation
model
Self. robot _ controller = robotorchestra () # robot band control
Self.vocal _ synthesizer = vocal synthesis () # singing synthesis module
Self. audience _ interaction = audience sensor () # audience interaction
perception

def run_concert(self, setlist):
for song in setlist:
# Phased execution: generating music score, robot playing, singing and
audience interaction.
score = self.music_model.generate_score(song)
self.robot_controller.perform(score)
self.vocal_synthesizer.sing(score.lyrics)
self.audience_interaction.detect_applause()
```

Second, the realization of key technologies

1. Multi-modal music generation and arrangement

Core model: A multi-modal model 37 similar to "Zidong Taichu" is used, which combines music understanding (MIDI analysis) and production ability.

Code example: generating the symphony version of "Jingle Bells"

```
from transformers import AutoModelForAudioGeneration
```

```
model = AutoModelForAudioGeneration.from_pretrained("Zidong-MusicGen")
Prompt = "Symphony version of Jingle Bells, including piano, violin, cello,
saxophone and French horn, with a cheerful rhythm"
score = model.generate(prompt, max_length=1000, temperature=0.9)
```

2. Robot band control

Hardware interface: the manipulator, pneumatic device and other actuators are controlled by ROS (Robot Operating System).

Motion control logic (taking piano robot as an example);

```
class PianoRobot:
def play_note(self, note, velocity):
# Control fingers to press keys through servo motor.
self.arm.move_to(note.position)
self.finger.press(velocity, duration=note.duration)
```

3. Multi-modal interactive command system

Command the robot: capture the beat through the visual sensor and adjust the band speed in real time.

```
class ConductorRobot:
def sync_tempo(self):
# Analysis of baton trajectory with OpenCV
_, baton_pose = cv2.VideoCapture(0).read()
tempo = self.pose_to_tempo(baton_pose)
Self.broadcast_tempo(tempo) # Broadcast the beat to the band through MQTT.
```

4. Singing synthesis and emotional expression

Speech model: using end-to-end speech synthesis similar to GPT-4o, adding emotional parameters (such as "passionate" or "soft");

```
vocal_model.generate_audio(
Text= "Friendship lasts forever, we used to wander around all day ...",
speaker_style={"emotion": "nostalgic", "vibrato": 0.7}
)
```

5. Audience interaction and real-time feedback

Sensor network: detect applause intensity through microphone array and camera, and dynamically adjust encore tracks;

```
if audience_interaction.applause_intensity > 0.8:
Self.run_concert(["Encore: ode to joy "])
```

Third, suggestions for developing tool chain

Fourth, the complete process example
Pre-training stage: training multi-modal generation model with massive symphony data.
Real-time performance stage:
Command the robot to capture the beat through LiDAR and generate the global timestamp.
The piano robot drives the mechanical fingers according to the music score, and the saxophone robot controls the airflow solenoid valve.
Singing synthesis: according to the lyrics, male and female voices are generated and mixed with musical instrument tracks.
Audience interaction: trigger lighting special effects and applause sampling cycle 4 through distributed sensors.

V. Challenges and Optimization Direction
Low delay requirement: it is necessary to optimize the reasoning speed of the model and use TensorRT quantization.
Multi-robot cooperation: using distributed consistency algorithm (such as Raft).
Safety redundancy: set the force feedback emergency stop for the mechanical arm.
Through the integration of the above technologies, a robot concert completely driven by AI can be realized. The complete code needs to be debugged according to the specific hardware interface. It is recommended to refer to the multimodal development framework and real-time interaction scheme mentioned in. ●
Complete program code about the concert held by artificial intelligence big model multi-model multi-modal general agent high intelligent robot, but a general idea and some code examples will help you build such a system.

Overview of system architecture

1. Music generation module: using music generation model (such as YuE) to generate music score according to the input theme or style.

2. Robot control module: through the robot control interface, the generated music score is converted into instructions for robot performance.

3. Multi-modal fusion module: integrating multi-modal information such as vision and hearing to realize the interaction between robot and environment.

4. Master control program: coordinate all modules and control the whole process of the concert.

Some code examples

```python
# Music Generation Example (using YuE

Program Code for Concert of Super Intelligent Robot with Large Model, Multi-model and Multi-mode General AgentGrand modèle multi-modèle multi-modèle intelligent robot universel ultra-intelligent code de programme de concert●
L'écriture d'instructions de jeu de robot est le processus de transformation des informations musicales en instructions de mouvement qu'un robot peut exécuter. Voici une idée générale et des étapes de mise en œuvre pour vous aider à écrire des instructions de jeu de robot:

Analyse de la partition musicale et conversion de paramètres
Les informations sur les notes de la partition (telles que la hauteur, le temps, l'intensité, etc.) sont analysées en paramètres de contrôle de jeu que le robot peut comprendre. Par exemple, pour un robot jouant du piano, il est nécessaire de convertir la hauteur et le temps de chaque note dans la position et la durée des touches correspondantes.

Conception logique de contrôle d'action
En fonction du type d’instrument joué par le robot, la logique de contrôle du mouvement est conçue. Par exemple, pour un robot piano, il est nécessaire de contrôler un bras robotique ou un aimant électrique pour appuyer sur la touche correspondante. Pour les robots à cordes, il est nécessaire de contrôler la position et la force de l'arc.

Jouer à la réception et le traitement du signal
Le robot doit être capable de recevoir le signal de jeu et d’entrer dans l’état de jeu correspondant en fonction du type de signal. Par exemple, lorsqu’un signal solo est reçu, le robot commence à jouer seul. Lorsque le signal d'ensemble est reçu, le robot entre dans un état d'attente et attend que les autres signaux continuent à jouer.

Plusieurs robots jouent ensemble
Dans une scène d'orchestre, plusieurs robots doivent jouer ensemble. Cela nécessite la conception d'un système de commande qui coordonne le rythme et la mélodie des différents robots pour assurer l'harmonie globale du jeu.

Exemples d'étapes de réalisation
```

Analyse de la partition : analyse les fichiers de partition (tels que les fichiers MIDI) en structures de données contenant des informations sur les notes.

Conversion de paramètres : les informations de la note sont converties en paramètres de contrôle du robot, tels que la position des touches, la durée et l'intensité.

Génération d'instructions de contrôle: Génération d'instructions de contrôle que le robot peut exécuter en fonction des paramètres de contrôle, tels que des signaux d'impulsion ou des instructions de commande de moteur.

Réception et traitement du signal: Écrire le module de réception du signal, traiter le signal de jeu et contrôler le robot dans l'état correspondant.

Contrôle de jeu collaboratif : dans une scène multi-robot, le système de contrôle principal est conçu pour coordonner les actions de jeu de chaque robot.

Grâce aux étapes ci-dessus, vous pouvez écrire les instructions de jeu correspondantes pour différents types de robots et réaliser leur jeu sur divers instruments. ● Pour réaliser un tel grand modèle multi-modèle d'intelligence artificielle, l'intelligence universelle multi-modèle du robot intelligent pour organiser le programme complet du code de concert est extrêmement complexe, impliquant l'intégration de la technologie dans de nombreux domaines, ce qui suit n'est qu'un exemple très simple de code Python conceptuel cadre (réaliste est loin d'être suffisant et nécessite une combinaison de beaucoup de matériel et d'algorithmes plus complexes, ici seulement une illustration), principalement impliquant la musique de lecture simulée et d'autres parties:

Le python

```
import playsound # Pour la lecture audio, cette bibliothèque doit être installée :
pip install playsound

# Définir une fonction pour jouer de la musique
Déf play_music(music_path) :
essayer :
playsound.playsound(music_path)
Excepté :
print(f"Impossible de lire la musique: {music_path}")

# En supposant que les chemins de fichiers audio de différents instruments et
chant
bell_path = "Tinker.mp3" # remplacé par le chemin du fichier réel
chemin_amitié = "amitié.mp3" # doit être remplacé par le chemin du fichier réel
piano_path = "fragment de piano .mp3" # remplacé par le chemin du fichier réel
violin_path = "fragment de violon .mp3" # remplacé par le chemin du fichier réel
```

cello_path = "fragment de violoncelle .mp3" # remplacé par le chemin du fichier réel
saxophone_path = "saxophone.mp3" # remplacé par le chemin du fichier réel
french_horn_path = "fragment de trompette française .mp3" # doit être remplacé par le chemin du fichier réel
male_singer_path = "fragment de chanteur masculin .mp3" # doit être remplacé par le chemin du fichier réel
female_singer_path = "fragment de la chanteuse .mp3" # remplacé par le chemin du fichier réel

Simulation d'un orchestre dirigeant battre (Ici un simple conseil d'impression)
Déf conduct():
print("Le chef d'orchestre commence à battre, le concert commence!"))

Simulation du jeu et du chant
Définir la performance():
Conduct()
print("Commencez à jouer la cloche")
play_music(belle_path)
print("Section du piano")
play_music(piano_path)
print("Section du violon")
play_music(violin_path)
print("Section du violoncelle")
play_music(cello_path)
print("Section du saxophone")
play_music(saxophone_path)
print("Section de la trompette française")
play_music(french_horn_path)
print("Section chanteuse masculine")
play_music(male_singer_path)
print("Section de la chanteuse")
play_music(female_singer_path)
print("Commencez à jouer l'Amitié pour toujours")
play_music(chemin de l'amitié)
print("Le concert est terminé, le public applaudit!"))

Si __name__ == "__main__" :
performance()
à

Le code ci-dessus n'est qu'une simulation de base, le concert de robot intelligent réel nécessite de nombreuses technologies et systèmes complexes tels que le contrôle du robot (y compris le contrôle du bras robotique de l'opération de l'instrument), l'interaction multimodale (comme l'identification visuelle des réactions du spectateur, etc.), la synthèse et le traitement audio (pour réaliser la synchronisation de différents instruments et chant, etc.), la prise de décision

intelligente (comme l'ajustement du jeu en fonction de la situation en direct, etc.).

● Pour réaliser des concerts réalisés en collaboration par des agents universels multimodaux et des bandes de robots, il est nécessaire d'intégrer les technologies de base telles que le grand modèle d'intelligence artificielle, l'interaction multimodale, la technologie audio et vidéo en temps réel et le contrôle du mouvement du robot. Voici un résumé de la conception du module de base et de la voie de mise en œuvre technique (le code complet doit être combiné avec une interface matérielle spécifique et un cadre de développement, le cadre technique et la logique de code clé sont fournis ici) :

à

I. Conception architecturale du système

à

```
# Exemple de structure globale
Système de concert :
Définir __init__(self) :
Initialisation du module de base
self.music_model = MultiModalMusicModel() # Modèle de génération de musique multimodale
self.robot_controller = RobotOrchestra() # Contrôle de bande
self.vocal_synthesizer = VocalSynthesis() # Module de synthèse de chant
self.audience_interaction = AudienceSensor() # perception de l'interaction du public

def run_concert(self, setlist) :
pour chanson dans setlist :
# Exécution par étapes: générer des partitions de musique robot jouer chant synthétiser l'interaction du public
score = self.music_model.generate_score(song)
self.robot_controller.perform(score)
self.vocal_synthesizer.sing(score.lyrics)
self.audience_interaction.detect_applause()
```

à

à

B. Réalisation des technologies clés

Génération et orchestration de musique multimodale

Modèle de base : Utilisez un modèle de grande taille multimodal 37 similaire à « Purple East Early », qui combine la compréhension musicale (analyse MIDI) avec des capacités de génération.

Exemple de code : Générer une version symphonique de la partition des cloches

à

```
from transformers import AutoModelForAudioGeneration

modèle = AutoModelForAudioGeneration.from_pretrained("Zidong-MusicGen")
prompt = "Version symphonique des clochers, avec piano, violon, violoncelle, saxophone, français, rythme joyeux"
score = model.generate(prompt, max_length=1000, température=0.9)
```

à
Contrôle des bandes de robots
Interface matérielle: Contrôle des bras robotiques, des dispositifs pneumatiques, etc. via ROS (système d'exploitation robotique).
Logique de contrôle de mouvement (à l'exemple d'un robot piano) :
à
```
Class PianoRobot :
def play_note(self, note, vitesse) :
# Contrôler les doigts par le servomoteur en appuyant sur la touche
self.arm.move_to(note.position)
self.finger.press(velocity, duration=note.duration)
```
à
Système de commandement interactif multimodal
Robot de commande : capture des battements avec des capteurs visuels pour ajuster la vitesse du groupe en temps réel.
à
```
Class ConductorRobot :
Déf sync_tempo(self) :
# Utiliser OpenCV pour analyser la trajectoire du mouvement de la barre de commande
_, baton_pose = cv2.VideoCapture(0).read()
temps = self.pose_to_tempo(baton_pose)
self.broadcast_tempo(tempo) # Transmet des battements au groupe via MQTT
```
à
Chant synthétique et expression émotionnelle
Modèle vocal : utilisez une synthèse vocale de bout en bout similaire à GPT-4o pour ajouter des paramètres émotionnels (comme « passionné » ou « doux ») :
à
```
Voix_modèle.generate_audio(
text="L'amitié dure depuis longtemps, nous errions toute la journée..."
speaker_style={"emotion": "nostalgic", "vibrato": 0.7}
)
```
à
Interaction du public et rétroaction en temps réel
Réseau de capteurs : l'intensité des applaudissements est détectée à l'aide d'un réseau de microphones et d'une caméra, l'ajustement dynamique des pistes d'Encore :
à
```
if audience_interaction.applause_intensity > 0.8 :
self.run_concert(["Encore: joie"])
```
à
à
Développement de la chaîne d'outils
à
à
à
à

à
à
Exemple de processus complet
Phase de pré-entraînement : entraîner un modèle de génération multimodale à l’aide de données symphoniques massives.
Phase de jeu en temps réel :
Le robot de commande capture les battements via le LiDAR pour générer un horodatage global.
Le robot piano entraîne les doigts mécaniques en fonction de la partition musicale et le saxophone contrôle la soupape électromagnétique du flux d'air.
Synthèse de chant : générer des voix masculine et féminine en fonction des paroles, mélangées avec des pistes d'instruments.
Interaction du spectateur : Déclenchement d'effets lumineux avec un cycle d'échantillonnage d'applaudissements par le biais d'un capteur distribué4.
à
V. Défis et orientations d’optimisation
Exigences de faible latence : la vitesse d'inférence du modèle doit être optimisée et quantifiée avec TensorRT.
Collaboration multi-robots : utilisez des algorithmes de cohérence distribués (par exemple, Raft).
Redondance de sécurité : Définissez un arrêt d'urgence pour la rétroaction de force du bras robotique.
Grâce à l'intégration de ces technologies, des concerts de robots entièrement alimentés par l'IA peuvent être réalisés. Le code complet doit être débogé en fonction des interfaces matérielles spécifiques, et le cadre de développement multimodal et les scénarios d'interaction en temps réel mentionnés dans la référence sont recommandés. ● Code de procédure complet sur l'intelligence artificielle à grande échelle Multi-modèle Multi-modèle Intelligence universelle Intelligent Robot pour organiser des concerts, mais une idée générale et des exemples de code partiel pour vous aider à construire un tel système.

Résumé de l'architecture système

Module de génération de musique: Utilisez des modèles de génération de musique (tels que YuE) pour générer des partitions musicales en fonction du thème ou du style d'entrée.

Le module de contrôle du robot: via l'interface de contrôle du robot, la partition générée est convertie en instructions pour le robot.

Module de fusion multimodale: intègre des informations multimodales telles que la vision, l'audition et d'autres, pour réaliser l'interaction entre le robot et l'environnement.

Programme de maîtrise: Coordonner les modules et contrôler le flux global du concert.

Exemple de code partiel

« Le python
Exemple de génération de musique (avec YuE)

.